

Automatic Generation of Perl and Python Extensions to C++ and Fortran Class Libraries

Craig Rasmussen

Advanced Computing Laboratory
Los Alamos National Laboratory

`rasmussn@lanl.gov`

ACTS Toolkit Workshop
September 2000

Credits

- **Craig Rasmussen** – Los Alamos National Laboratory, rasmussn@lanl.gov
- **Reid Rivenburgh** – Los Alamos National Laboratory, reid@lanl.gov
- **Kathleen Lindlan** – University of Oregon, klindlan@cs.oregon.edu
- **Bernd Mohr** – Central Institute for Applied Mathematics, Jülich, Germany, B.Mohr@fz-juelich.de
- **Pete Beckman** – TurboLinux, beckman@turbolabs.com

Introduction

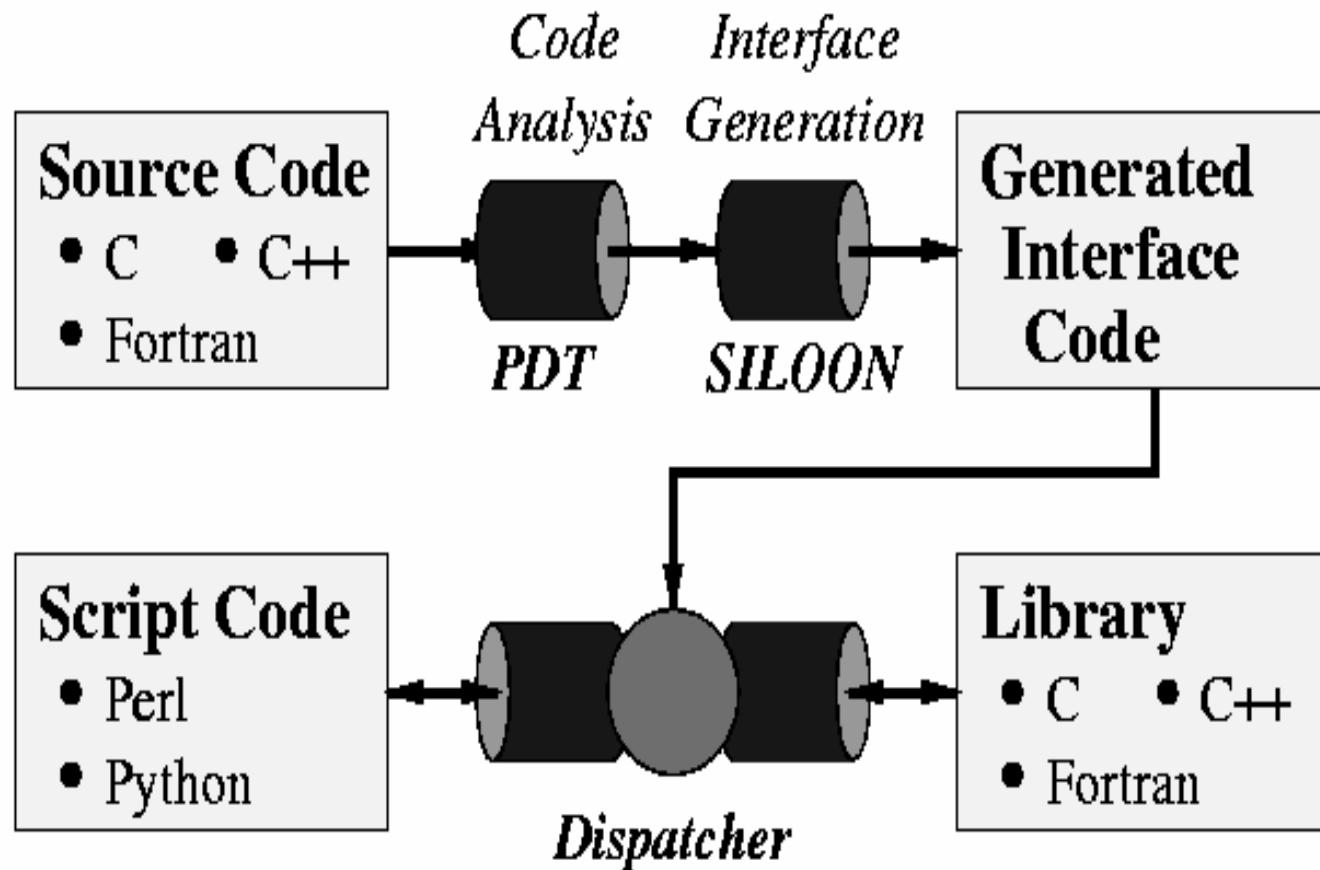
- What
 - Interface scripting languages with C++/Fortran libraries
- Why
 - Taming of the beasts (C++ and Fortran)
- Why not SWIG
 - C++ templates
 - Fortran 90

Why Useful

- Los Alamos environment
 - a bunch of physicists
 - letting physicists loose on C++?
 - legacy Fortran and bleeding C++
- Power, flexibility, and rapid development offered by scripting languages combined with...
- Speed and availability of existing C++ and Fortran libraries

Supported Languages

- Scripting languages (client)
 - Perl
 - Python
- Compiled languages (compute server)
 - C
 - C++
 - Fortran



How It Works [1]

- Program Database Toolkit
 - <http://www.cs.uoregon.edu/research/paracomp/pdtoolkit/>
 - C++ parsing
 - EDG front end; commercial, high-quality C++ parser
 - Generates an intermediate language file
 - C++ analysis
 - Generates Program Database (PDB) file; structured, mostly complete description of program
 - DUCTAPE
 - Provides object oriented, C++ interface to PDB file
 - Used to generate SILOON's bridging code

How It Works [2]

- SILOON bridging code
 - scripting languages: wrappers (proxy pattern), siloon_python.cc, siloon_perl.xs (uses Perl XSUB)
 - Active Messages
 - function id and parameters marshaled into a buffer
 - C++: siloon_register.cc, siloon_execute.cc
 - Mangling used to support difficult C++ constructs, such as overloaded functions:
 - Complex &Complex::Complex(int, int) → Complex_I_I

Example [1]

example.cpp:

```
template<class T>
void func(T &t)
{
    std::cout << t << endl;
}

class Complex {
public:
    Complex() { real_ = 0; imag_ = 0; }
    Complex(int r, int i) { real_ = r; imag_ = i; }
    ~Complex(void) { }

    static float norm(const Complex& c)
        { return sqrt(r*r + float(i*i)); }

    void operator+(int i) { real_ += i; }

private:
    int real_;      // real part
    int imag_;      // imaginary part
};
```

Example [2]

- Basic steps to create a SILOON module:

```
[1]% siloon-init example --perl --python  
[2]% cd siloon-example  
[3]% vi user.defs  
[4]% emacs prototypes.doinclude  
[5]% make interface  
[6]% make shared
```

Example [3]

user.defs:

```
# User source files.  Ex: /home/user/project/library.cc
SILOON_USER_SOURCES=../example.cpp

# Any include paths and variables needed during the siloon-parse
# stage
SILOON_PARSE_INCLUDES=-I../..

# Any include paths.  Ex: -I/home/user/project/include -DDEBUG
SILOON_USER_INCLUDES=${SILOON_PARSE_INCLUDES}

# Any library paths and names.  Ex: -L/home/user/project/lib -lm
SILOON_USER_LIBRARIES=

# Any object files.  Ex: /home/user/project/library.o
SILOON_USER_OBJECTS=../.../example.o
```

Example [4]

prototypes.doinclude:

```
# Copy lines here from prototypes.excluded to force SILOON to
# include a prototype in the generated interface.
# Or, add ":filename" or ":dir" to include all prototypes in
# a given file or directory. An ending ":" means to do so
# recursively in the case of a directory.
:/home/siloon/example:
"func"           "void func<int >(int)"
"Complex"        "Complex &Complex::Complex()"
"ComplexInt"     "Complex &Complex::Complex(int, int)"
"delete_Complex" "void Complex::~Complex()"
"norm"           "float Complex::norm(const Complex &)"
"op_plus"         "void Complex::operator+(int)"
```

Example [5]

example-test.pl:

```
BEGIN {
    push(@INC, "./siloon-example/perl/blib/arch/auto/example_siloon");
    push(@INC, "./siloon-example/perl");
};

use ExtUtils::testlib;
use example;

$complex = ComplexInt->new(3,4);
$norm = $complex->norm($complex);
func($norm);

# Perl garbage collection automatically calls Complex destructor.

exit(0);
```

C++ Support

- SILOON provides support for most C++ features, including:
 - classes
 - virtual and static member functions
 - constructors and destructors
 - overloaded functions
 - default function arguments

C++ Support

- Features (cont.)
 - references
 - enumerations
 - templates
 - typedefs
 - Standard Template Library
 - operators

Fortran

- Stay tuned....

Issues [1]

- Pointers
 - memory allocated on server
- Arrays
 - passed by pointer
 - special functions to read and write
- References
 - passed by pointer
- Strings
 - passed by value

Issues [2]

- Name mangling
 - templates
 - operators
 - overloaded functions
- In/out parameters
 - problem for references (int &)

Frameworks and Users

- Viz
- POOMA
- PAWS
- MTL
- AOL?

Future Plans

- Better Fortran support
- Java support
- CORBA IDL
- Objective C
- Namespaces
- Multiple modules
- In/out parameter handling
- Open source C++ parser interface

Links

- SILOON home page:
 - <http://www.acl.lanl.gov/siloon>
 - siloon-team@acl.lanl.gov